공간 데이터의 특성을 고려한 저장 관리자

김종훈[†] · 정현민[††] · 장성인[†††] · 정미영[††††]

요 약

공간 데이터베이스 시스템에서의 공간 데이터에 대한 관리는 전체적인 시스템의 성능을 좌우하며 따라서 이를 위한 처리비용을 최소로 줄이는 기법이 요구된다. 하지만 공간 데이터는 기존의 멀티미디어 데이터와는 달리 그 특성상 레이어 단위로는 데이터의 크기가 비교적 비슷하고, 개개의 데이터 단위로는 그 크기가 수 바이트에서 수 테라(Tera) 바이트까지 다양하다는 특성을 가지고 있어 최소 저장 단위가 한 페이지 이상인 EXODUS나 Starburst의 대용량 관리 모듈이나 BLOB등의 멀티미디어 데이터 저장 관리 기법을 그대로 적용할 경우 디스크의 공간 낭비 및 시스템의 성능 저하라는 문제를 가지게 된다. 따라서 본 논문에서는 이러한 공간 데이터의 특성을 고려한 공간 데이터 저장 기법을 제시하고자 한다.

Storage Manager Considering Spatial Data Characteristics

Kim Jong Hoon[†], Junh Hyun Meen[††], Jang Sung In[†††] and Jeong Mi Young[††††]

ABSTRACT

As total system performance depends on spatial data management in spatial database system, low cost method is required. However, spatial data have many characters that are different from multimedia data, data size is almost similar by layer and variable from few bytes to tera bytes. So large data manager of EXODUS and Starburst and BLOB etc, make problems that is many Disk I/O and Disk space waste. This paper proposes new storage method for spatial data considering spatial data characteristics.

1. 서 론

컴퓨팅 환경의 발전으로 인하여 기존의 정형화된 텍스트 데이터뿐만 아니라 비정형화된 멀티미디어 데이터에 대한 사용자의 처리 요구가 증대함에 따라 데이터베이스 시스템 또한 이러한 요구사항을 반영할 수 있도록 확장되고 있다. 이러한 비정형 데이터 중에서도 공간 데이터 타입(Spatial Data Type)을 지원할 수 있도록 확장된 데이터베이스 시스템을 공간 데이터베이스 시스템(Spatial Database System)이라 하며[1], 고성능의 공간 데이터베이스 시스템을 위해서는 공간 데이터를 효과적으로 저장하고 관리하는 공간 저장 관리자(Spatial Storage Manager)의 개발이 필수적으로 요구된다.

이러한 대용량의 멀티미디어 데이터를 저장 및 관리하기 위한 많은 연구가 진행되고 있으며, 국외의 가장 대표적인 사례로 EXODUS 저장 관리자[2,3]와 Starburst 롱 필드 관리자[4]의 대용량 관리 모듈을 들 수 있다. EXODUS의 저장 관리자는 기본적으로 B^{+}-Tree 구조의 인덱스를 이용하여 임의 위치 접근이 용이하도록 설계되었고, Starburst 롱 필드 관리자는 빠른 순차 접근을 위해 할당 정책으로 버디 시스템(Buddy System)을 채택하였다.

그러나 공간 데이터는 그 특성이 기존의 멀티미디어 데이터와는 달리 레이어(Layer) 단위로는 데이터의 크기가 비교적 균일하지만 개개의 데이터 크기는 수 바이트에서 수 테라 바이트까지 다양하다는 데이터 특성을 가지고 있어 기존에 제시된 멀티미디어 데이터 저장 관리 기법을 그대로 사용할 경우 다음과 같은 문제가 발생한다.

[†] 정회원, (주)케이지아이 대표이사
[††] 한국통신 가입자망연구소 무선망설계연구실장
[†††] 한국통신 가입자망연구소 선임연구원
[††††] 한국통신 가입자망연구소 전임연구원

첫째, 기존의 대용량의 멀티미디어 데이터의 저장 관리 기법은 데이터 저장 단위가 한 페이지 이상이기 때문에 객체의 크기가 한 페이지보다 작은 공간 데이터를 관리할 경우 페이지에 대부분의 공간이 낭비가 되므로 데이터 크기에 비해 디스크 입출력이 많이 발생하여 시스템의 성능이 저하되는 문제가 발생한다.

둘째, 공간 데이터의 접근은 고비용의 공간연산을 수반하기 때문에 객체의 최소경계사각형(MBR)을 이용하여 여과(Filtering)를 수행하고, 여과된 후보 객체에 대해 여과정제(Filtering Refinement)를 수행하게 된다. 따라서 공간 데이터에 대한 여과 정보를 담고있는 헤더(Header)를 실제 공간 데이터와 같이 저장하게 되면 필요한 데이터에 비해 많은 디스크 입출력이 발생하여 성능이 저하되는 단점이 있다.

셋째, 빠른 서비스가 요구되는 웹(Web)과 같은 환경에서는 자주 접근되는 데이터에 대해 캐싱을 하게 되는데, 기존의 대용량 데이터 저장기법은 정형화된 데이터에 대해서는 버퍼를 이용하여 시스템의 효율을 높이지만 대용량 데이터에 대해서는 버퍼링을 지원하지 않기 때문에 응용 프로그램이 별도의 캐싱을 해야 된다는 어려움이 발생한다.

넷째, 멀티미디어 데이터의 회복은 정형화된 데이터에 대해서 로깅(Logging)에 기반 한 고장 회복 기법을 사용하고 대용량 데이터에 대해서는 데이터의 특성상 많은 디스크 입출력을 발생시키는 로깅 방법보다는 쉐도우 페이지(Shadow Page)기법을 이용한다[3,4]. 그러나 갱신 크기에 무관하게 페이지를 쉐도우 시키기 때문에 공간 데이터의 크기가 작을 경우에는 많은 디스크 입출력을 발생시켜 회복 비용이 커지는 문제점이 있다.

따라서 본 논문에서는 이러한 문제점을 해결하기 위해 공간 데이터의 특성을 고려한 저장 관리자를 제안한다. 제안된 기법은 공간 연산 부하 및 공간 인덱스(Spatial Index) 생성 부하를 줄이기 위해 여과를 위한 공간 데이터의 헤더 정보를 비공간 데이터 페이지에 함께 저장시키고, 공간 데이터의 실제 점 리스트는 그 크기가 일정 규모보다 작을 때는 저장 공간이 낭비되는 것을 막기 위해 비공간 데이터와 같은 페이지에 저장하고, 일정 크기보다 큰 경우에는 대용량 데이터 저장방법인 BLOB 형식으로 데이터를 저장하는 이중 구조를 제안한다. 이를 위해 동시성 제어 및 고장회복 기법도 엄정 2단계 잠금 규약(Strict Two Phase Locking Protocol)을 기반으로 데이터의 크기에 따라 공간 데이터의 크기가 작을 경우에는 로깅 기법을 사용하고, 크기가 큰 경우에는 쉐도우 페이지 기법을 병행으로 사용하는 방법을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 대용량 데이터의 특성과 기존 대용량 데이터 저장 관리 기법과 회복 기법에 대해 고찰하고, 3장에서는 본 논문에서 제안한 공간 데이터 저장 관리 기법에 대해서 기술한다. 4장에서는 공간 저장 관리자를 설계 및 구현 방법을 기술하고 이를 실험 평가하며, 끝으로 5장에서 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 멀티미디어 데이터와 공간 데이터의 특성

최근 CPU 속도향상, 메모리의 고집적화, 그리고 디스크의 대용량화에 따라 실생활에 보다 근접된 그래픽, 이미지, 애니메이션, 오디오, 비디오 등의 멀티미디어 데이터에 대한 사용자의 요구가 급증하고 있다. 이러한 멀티미디어 데이터의 특성은 크기가 매우 크다는 것인데, 예를 들어 400dpi 비트맵 이미지의 경우 약 2메가 바이트, 5분 가량의 오디오의 경우 약 50메가 바이트, 그리고 초당 12 프레임으로 구성된 5분 가량의 고품질 비디오인 경우 약 850메가 바이트의 저장 공간이 각각 요구된다[10,11]. 이것은 대략 수 바이트 크기의 저장 공간을 요구하는 기존의 수치, 문자 데이터와 비교하여 엄청난 차이를 보이는 것이다. 이런 특성 외에도 멀티미디어 데이터는 대부분 접근 특성상 순차접근이 일반적이며 갱신 연산보다는 읽기 연산이 대부분이라는 특성을 가지고 있다.

공간 데이터 또한 멀티미디어 데이터와 같이 대용량이고 접근패턴 또한 순차접근이라는 점에서는 같은 특성을 가지고 있다. 그러나 일반적인 멀티미디어 데이터와 달리 데이터의 크기가 수 바이트에서 테라바이트까지 매우 가변적이다. 또한 지도를 구성할 때, 속성이 비슷한 공간 객체를 레이어 단위로 저장하게 되는데 동일한 레이어에 속한 객체의 크기는 비교적 균일하다는 특성을 갖게된다. 서울시 지도의 각 레이어 별로 데이터의 크기 분포도를 보이고 있는 그림 1, 그림 2를 보면 각 레이어에 따라 데이터의
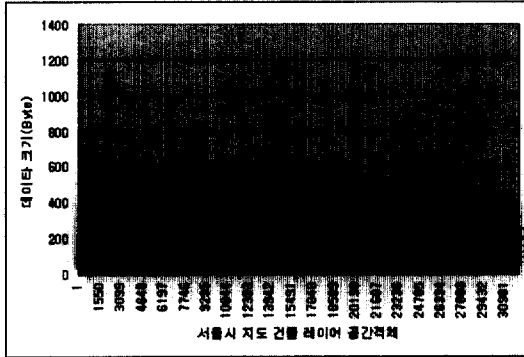
그림 1. 서울시 건물 레이어 공간객체 크기 분포도
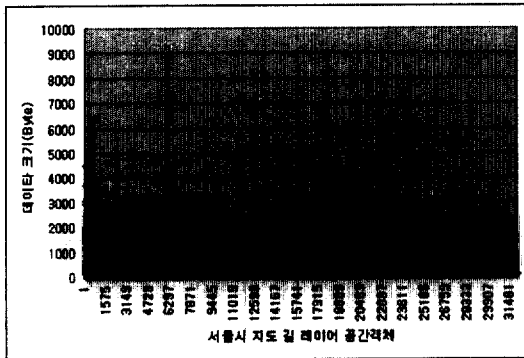
그림 2. 서울시 길 레이어 공간객체 크기 분포도

크기가 비교적 균일하게 나타남을 알 수 있다.

이러한 공간 데이터는 공간질의 수행시 Disjoint, Intersect, Contain, Equal, Touch, Within, Overlap, Cross 등과 같은 공간연산을 수행하게 되는데, 연산 자체가 고비용이고, 또한 공간 데이터가 대용량 데이터일 경우에는 많은 디스크의 입출력이 발생하여 시스템의 전체적인 성능을 저하시킨다. 따라서 이러한 고비용의 공간연산을 절감하기 위해서 일반 질의 과정과는 달리 다음 두 단계를 거쳐 수행하게 된다.

- 여과단계: 질의 사각형과 객체의 최소경계사각형을 이용하여 질의 조건에 만족하는 객체를 추출한다.
- 정제단계: 여과단계의 결과인 후보객체들에 대하여 질의에 만족하는 객체를 찾는다.

일반적으로 여과단계에서는 R-Tree와 같은 공간 인덱스를 사용하게 되는데 R-Tree는 관리의 간편함과 연산의 간단함 때문에 공간 데이터에 대해 가장 일반적으로 채택되고 있는 공간 색인 구조로서 해당 객체의 최소경계사각형(MBR)을 이용하여 여과과정을 수행한다[5]. 이러한 여과과정을 효과적으로 수행하기 위해서는 별도의 R-Tree나 공간객체의 헤더 정보 접근이 요구된다. 따라서 여과과정에 필요한 데이터를 효율적으로 저장 및 관리해야 한다. 이러한 여과단계에서는 정확한 질의결과가 아닌 많은 후보 객체가 결과로 생성되게 되는데 정확한 결과를 얻기 위해 각 후보객체들에 대한 실제 데이터를 디스크로부터 읽어서 질의 조건에 만족하는지 확인하는 정제 과정을 거쳐야 한다.

2.2 기존 대용량 데이터 저장 관리 기법

현재 대용량 데이터를 처리하기 위한 많은 연구가 진행되어 왔으며, 그중 가장 대표적인 예로 Informix, Sybase등의 상용 DBMS에서 주로 채택하고 있는 BLOB[5], 그리고 EXODUS[2,3]와 Starburst[4]의 대용량 데이터 관리 모듈이 있다. BLOB은 그림 3처럼 페이지들의 연결 리스트의 형태로 데이터는 첫 페이지로부터 빈 공간 없이 차례로 저장되며, 마지막 페이지에서는 데이터를 저장하고 남은 빈 공간이 생길 수 있다.

EXODUS의 저장 관리자는 기본적으로 그림 4처럼 B^+-Tree구조의 인덱스를 이용하여 임의 위치 접근(Random Access)이 용이하도록 설계되었으나, 할당된 대용량 데이터들이 디스크에 분산되어 저장되므로 순차접근(Sequential Access)에 대한 성능이 떨어진다.

이와는 대조적으로 Starburst의 롱 필드관리자는 그림 5처럼 빠른 순차접근을 위해 할당 정책으로 버디 시스템을 채택하였다. 그러나 대용량 데이터에 대해 삽입, 삭제 연산을 수행할 때 많은 디스크 입출력을 발생시키는 단점이 있다.

위에서 제시된 기법들은 데이터가 항상 대용량 데이터인 경우만을 고려하여 설계되어 공간 데이터와 같이 데이터의 크기가 매우 가변적일 경우 디스크 공간 낭비 및 이로 인한 빈번한 디스크 입출력의 발생으로 시스템의 성능이 저하된다는 단점이 있다.
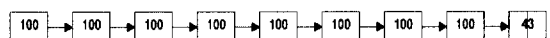
그림 3. 연결 리스트를 이용한 BLOB 관리 기법
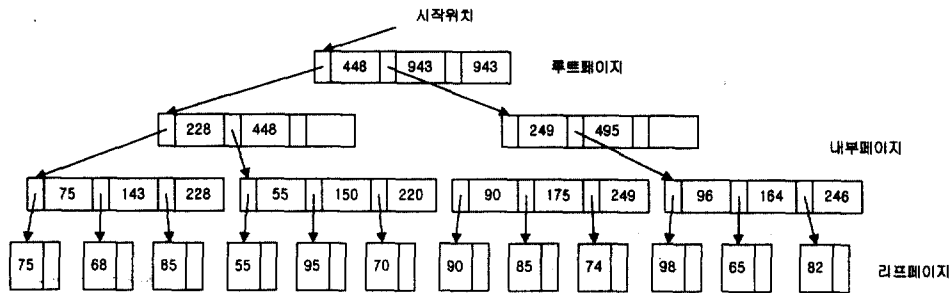
그림 4. Exodus 에서의 대형 객체 관리 기법
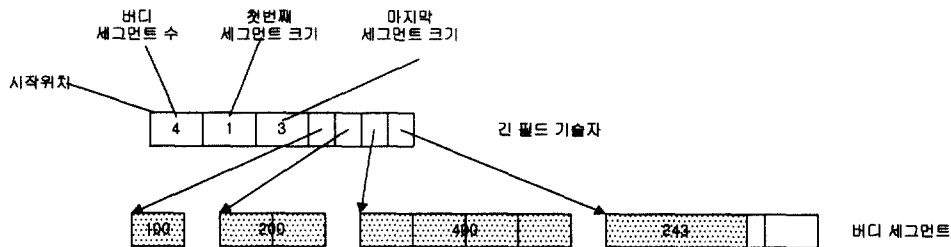
그림 5. Starburst 롱(Long) 필드관리자

2.3 회복을 위한 쉐도우 페이지 기법

대용량 데이터 관리 모듈들은 회복을 위해 대용량 데이터의 특성상 많은 디스크 입출력을 발생시키는 로깅 방법보다는 전체 세그먼트를 쉐도우 시키는 방법을 취하고 있다. 쉐도우 페이지방법은 트랜잭션이 페이지를 갱신하게 되면 새로운 페이지를 할당하고 그 페이지에 원래의 데이터를 복사하고 기존의 페이지에 새로운 데이터를 복사한다. 나중에 트랜잭션이 완료(Commit)하게되면 기존의 데이터가 기록된 쉐도우 페이지는 버려지게 되고, 만약 Abort되면 쉐도우 페이지의 데이터를 원래 데이터로 복구한다. 그러나 쉐도우 페이지 기법은 데이터의 갱신크기에 관계없이 페이지 단위로 회복을 수행하기 때문에 데이터의 크기가 페이지의 크기와 큰 차이가 있을 때에는 기존의 로깅방법보다 성능이 떨어진다는 단점이 있다.

3. 공간 데이터 특성을 고려한 공간 데이터 저장 관리 기법

3.1 공간 데이터의 크기를 고려한 저장 기법

공간 데이터는 크기가 몇 바이트에서 테라까지 다양하기 때문에 최소 저장 단위가 한 페이지 이상인 기존의 대용량 데이터 저장 관리기법을 이용하면 공간 데이터의 크기가 작을 시에는 시스템의 성능이 저하되는 문제가 발생한다. 왜냐하면 보통 대용량 데이터에 대해서 버퍼를 지원하지 않고, 또한 실제 데이터 크기에 비해 접근해야 되는 페이지의 개수가 많아지기 때문이다. 이 문제를 해결하기 위해서는 크기에 따라 데이터 저장기법을 달리해야 한다.

본 논문에서는 디스크 입출력 비용을 최소로 줄일 수 있는 공간 데이터 저장기법을 제시한다. 제안된 기법은 데이터의 크기에 따라 데이터의 크기가 페이지 크기보다 크면 데이터를 기존의 대용량 데이터 기법의 하나인 BLOB으로 저장하고 크기가 작으면 데이터를 속성데이터와 같은 페이지에 저장한다. 이렇게 함으로써 크기가 작은 데이터에 대해서 공간낭비의 문제를 해결할 수 있다. 또한 대용량데이터에 대해서는 버퍼를 지원하지 않고, 스몰 데이터에 대해서만 버퍼를 지원하기 때문에 자주 참조되는 크기가 작은 데이터에 대해서는 버퍼링도 기대할 수 있다. 반면에 공간 데이터가 크기가 클 때에는 대용량 데이터 기법인 BLOB으로 저장함으로써 무제한의 크기를 지원하고, 저장 공간 이용률이 높으며, 연속 액세스 효율을 높일 수 있다.

제안기법을 이용하기 위해서는 기존의 레코드 구조를 변경하여야 한다. 그래서 레코드의 구조는 먼저

현재 레코드의 공간 데이터가 비공간 데이터와 같이 저장이 되어있는지 아니면 BLOB으로 저장되었는지 대한 플래그(Flag)가 있어야 한다. 만약 비공간 데이터와 같은 레코드에 공간 데이터가 들어간다면 공간 데이터의 저장되는 위치가 필요하고, 공간 데이터가 대용량이어서 따로 BLOB에 저장되어 있다면 BLOB의 첫 번째 페이지 식별자(ID)가 필요하다. 이를 반영한 그림은 그림 6과 같다.
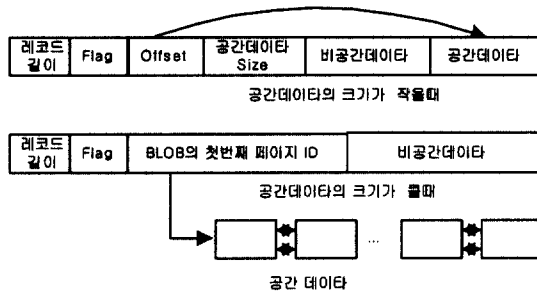
그림 6. 공간 데이터 크기를 고려한 레코드의 구조

3.2 공간 데이터의 접근 패턴의 고려

사용자의 질의는 보통 공간 데이터와 비공간 데이터에 동시에 이루어지기 때문에 공간테이블(Spatial Table)은 비공간 데이터와 공간 데이터가 따로 저장이 되면, 공간 데이터만으로는 처리할 수 없어서, 고비용의 조인연산이 필연적으로 발생하기 때문에 함께 저장된다. 그러나 공간 데이터에 대한 공간질의는 고비용의 공간연산을 필요로 한다. 이런 고비용의 공간연산 비용을 줄이기 위해 보통 여과를 수행한다. 이런 여과과정을 수행하기 위해서 객체의 최소경계사각형을 이용하여 R-Tree를 만들거나 R-Tree가 없으면 공간 객체에 대한 헤더정보가 필요하다. 또한 R-Tree를 생성할 때에도 객체의 헤더정보중의 하나인 객체의 최소경계사각형을 참조하기 때문에 공간 데이터의 헤더정보를 공간 데이터의 점리스트와 동일하게 저장하게 되면 공간 데이터에 대한 여과과정이나 R-Tree 구성시 많은 디스크 입출력이 발생하는 문제점이 발생한다. 이러한 문제점을 해결하기 위해서는 그림 7처럼 구성된 공간 데이터의 헤더와 실제 점리스트를 다르게 저장할 필요가 있다.

점갯수	객체 Type	최소경계사각형	점리스트

그림 7. 공간 데이터 구조

따라서 본 논문에서는 공간 데이터를 공간 데이터 헤더와 실제 점 리스트를 분할하여, 헤더데이터는 비공간 데이터의 레코드에 항상 같이 저장하고 실제 데이터인 점리스트를 크기에 따라 저장한다. 이를 반영하면 레코드의 구조는 그림 8과 같이 공간 데이터의 헤더는 항상 비공간 레코드와 함께 저장이 되고 공간 데이터의 점리스트는 크기에 따라서 작으면 비공간 데이터 레코드와 함께 저장이 되고, 크면 BLOB 형식으로 저장이 되고 BLOB의 첫 번째 페이지 ID만 비공간 데이터 레코드에 기록이 된다.

이렇게 저장된 공간 데이터는 커서를 이용하여 접근이 된다. 커서의 종류로는 보통커서와 색인커서가 있다. 그림 9처럼 보통커서로서 공간 데이터를 접근한다면, 먼저 원하는 비공간 데이터 레코드를 찾은 후에, 첫 번째 필드에 저장된 공간 데이터의 필드를 크기에 따라 작으면 비공간 데이터 레코드에서 접근하고, 크면 비공간 레코드에 저장된 BLOB의 첫 번째
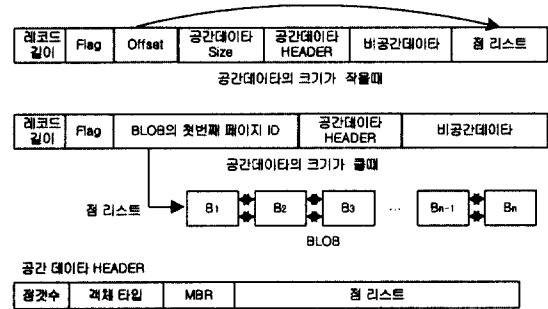
그림 8. 공간 데이터 접근패턴을 고려한 레코드의 구조
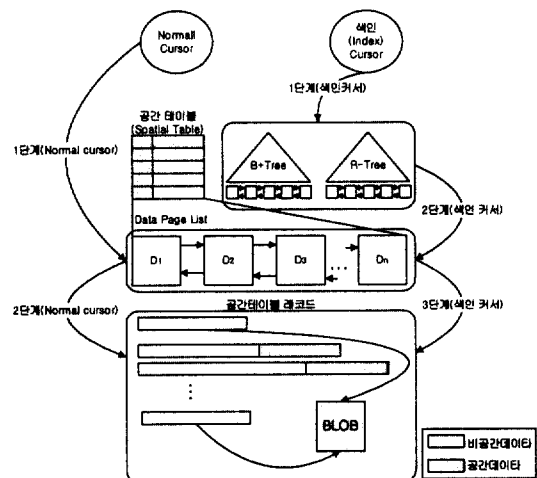
그림 9. 공간 데이터 접근 단계

페이지 식별자를 이용하여 공간 데이터를 접근한다. 만약 커서가 색인커서이면 현재 색인이 가리키는 레코드를 읽어들이면 된다. 여기서 색인이 가리키는 레코드는 항상 비공간 데이터의 레코드이다. 따라서 사용자는 항상 공간 데이터를 접근할 때 비공간 데이터 레코드를 먼저 접근하고 다음으로 공간 데이터를 접근한다. 이렇게 데이터의 접근방향이 일정하기 때문에 따로 공간 데이터에 대해 잠금을 요청할 필요가 없다. 따라서 잠금의 부하가 적으며, 또한 공간 데이터가 비공간 데이터 레코드를 가리키는 쌍방향의 링크가 필요가 없다.

3.3 공간 데이터의 동시성 제어 및 회복

제안기법은 ARIES를 사용하는 시스템을 기반으로 한다. 그러나 대용량의 데이터에 대해서 동일하게 ARIES의 회복의 근간인 로깅기법을 이용하면 디스크 공간 및 입출력이 많이 필요하여 시스템의 성능이 저하되어 대용량 데이터의 회복 방법으로는 입출력 비용을 줄일 수 있는 쉐도우 방법을 적용 바람직하다. 따라서, 본 논문에서는 한 페이지보다 작은 데이터에 대해서는 ARIES에 기반 한 로깅 회복기법을 적용하고 한 페이지보다 큰 대용량 데이터 대해서는 쉐도우 페이지 기법을 적용한다.

ARIES는 동시성 제어를 위해 잠금 기법을 이용하는데, 공간 데이터가 따로 저장되더라도 공간 데이터에 대한 연산은 먼저 연산대상이 될 비공간 레코드를 찾고 비공간 레코드에 있는 공간 데이터에 대해 연산이 수행되기 때문에, 비공간 데이터 레코드에 대해서만 잠금을 요청하면 된다. 로깅기법은 ARIES와 동일하기 때문에 언급하지 않고 공간 데이터가 대용량 데이터일 경우에만 언급을 하겠다. 대용량의 공간 데이터의 갱신연산은 앞에서 제시된 쉐도우 페이지 기법을 약간 변형하였다. 첫 번째로, 갱신(Update)연산은 기존의 데이터는 그대로 두고 새로운 데이터를 새로 할당된 BLOB에 기록하고 비공간 데이터 레코드에 새로운 BLOB의 첫번째 페이지 식별자를 기록한다. 그리고 갱신 트랜잭션이 완료할 때 기존의 BLOB 데이터를 삭제한다. 갱신연산을 수행할 때 BLOB의 삭제 연산을 트랜잭션의 완료시점까지 미루기 위하여 트랜잭션의 Pending 리스트라는 자료구조를 이용한다. 여기서 트랜잭션의 Pending 리스트는 각각의 트랜잭션들에 대해서 트랜잭션의 완료시점까지 지연시켜야 할 연산들을 가지고 있다. 이렇게 트랜잭션 Pending리스트에 추가하여 연산을 지연시키는 것은 BLOB에 속해있는 페이지가 반환되어 다른 데이터가 쓰여지게 된다면, 이전 데이터가 유실되어 현재 트랜잭션을 취소시킬 수 없기 때문이다. 두 번째로, 삽입연산은 BLOB 데이터의 삽입을 수행하고, 나중에 Rollback이나 Undo를 수행하게 되면 BLOB에 속해 있는 페이지를 반환하면 된다. 마지막으로 삭제는 갱신연산과 마찬가지로 BLOB에 속해있는 페이지를 즉시 반환하지 않고 트랜잭션의 Commit시점까지 지연시키기 위해서 트랜잭션의 Pending리스트에 BLOB의 삭제연산 추가한다.

4. 공간 데이터 관리자의 설계 및 구현

4.1 공간 저장 관리자의 구조

저장 관리자 레벨에서 공간 데이터 타입과 공간 데이터에 대한 연산 및 색인을 지원하는 것으로 다음 그림 10과 같은 구조를 갖는다.

본 저장관리자는 디스크 관리자 레벨에서는 대용량 데이터를 지원하기 위해 64비트의 주소를 페이지 식별자로 사용하여 각 볼륨(Volume)당 최대 4테라까지 지원하고 있다. 버퍼 관리자 레벨에서는 객체의 헤더정보의 신속한 접근을 위해 따로 객체 관리자에서 관리하며, 버퍼 관리자는 버퍼정책으로 Steal, No Force 정책을 채택하고 있다. 레코드 관리자 레벨에서는 공간 데이터의 지원을 위해 기본 데이터 타입 외에도 공간 데이터 타입이 추가되었다. 또한 레코드 관리자 레벨에서 공간 데이터에 대한 연산이 지원되며 색인으로는 B^+-Tree외에 공간색인으로 R-Tree를 지원하고 있다. 트랜잭션 레벨에서는 트랜잭션의 4단계 격리 래벨을 지원하고 있으며 동시성 제어를 위해 잠금기법을 이용하고 회복을 위해 ARIES와 쉐도우 페이지 기법을 이용한다.

4.2 공간 데이터 저장을 위한 자료 구조

공간 데이터는 공간테이블(Spatial Table)에 비공간 데이터와 함께 저장이 된다. 공간 테이블은 크게 한 페이지보다 작은 데이터가 들어가는 데이터 페이지와, 신속한 삽입연산을 위해 각 데이터 페이지의 식별자와 데이터 페이지에 남아있는 용량에 대한 정보를 가지고 있는 공간 사용 페이지(Free Info Page)
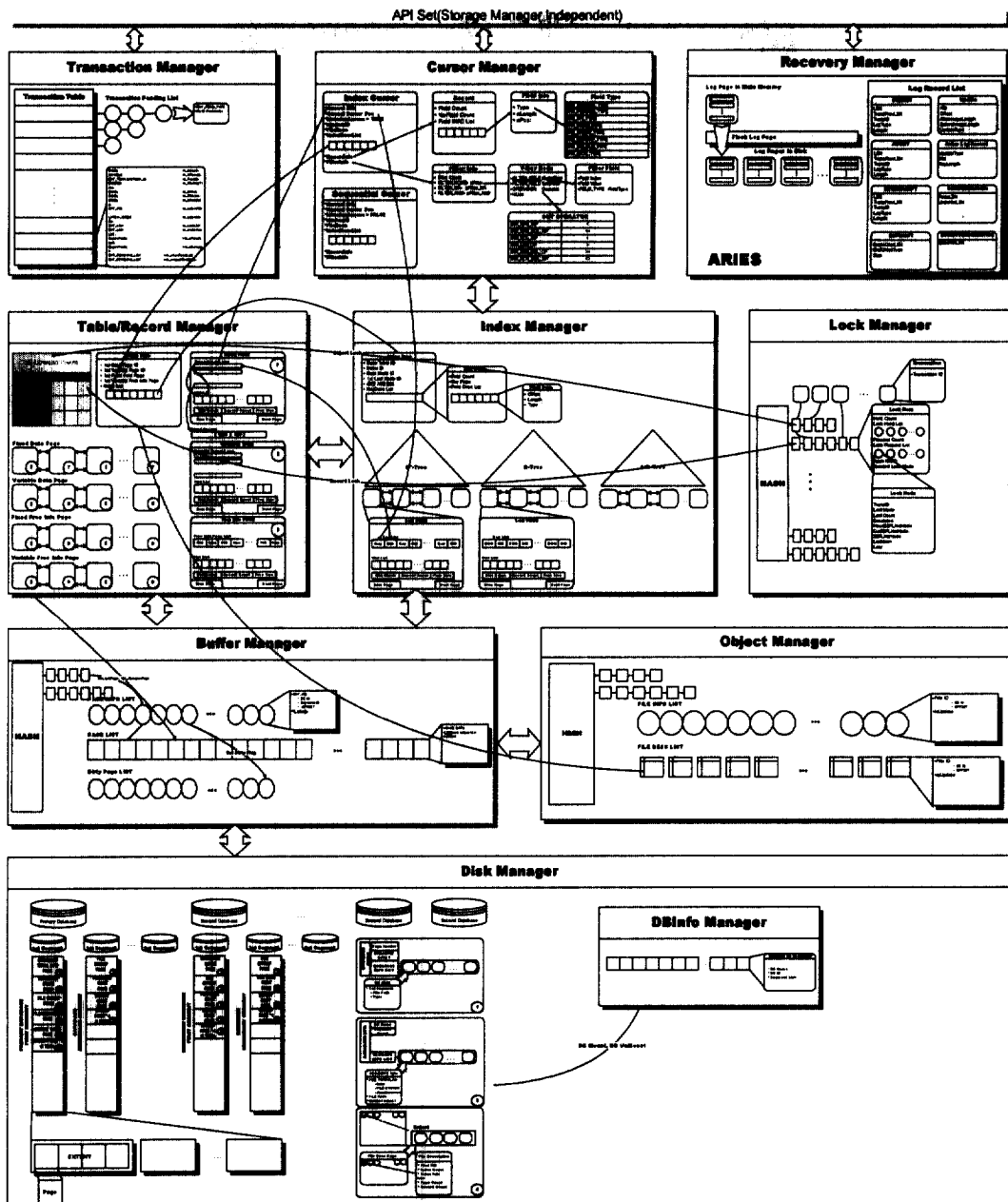
그림 10. 공간 저장 관리자의 구조

로 구성이 된다. 데이터 페이지에는 한 페이지보다 크기가 작은 레코드가 저장이 된다. 그리고 본 저장 관리자는 질의 처리기의 시스템 카탈로그와 중복되는 데이터로 인한 공간낭비 및 불필요한 디스크 입출력을 줄이기 위해 저장관리자 자체적으로 테이블의 필드 정보와 같은 자세한 테이블 정보를 가지고 있지 않다. 그러나 본 논문에서 제안된 기법은 공간 데이터의 크기에 데이터의 저장위치를 달리하기 때문에 공간 데이터가 저장되는 필드에 대한 정보가 필요하다. 이러한 문제를 해결하기 위해서 본 논문에서는 테이블의 종류가 공간 테이블이면 레코드의 첫 번째 필드는 공간 데이터가 저장되는 위치로 고정했다. 그리고 데이터 페이지에는 한 페이지보다 작은 레코드가 저장이 되기 때문에 대용량 데이터를 위해서 공간

데이터가 페이지 크기보다 클 때는 대용량 데이터 저장기법중의 하나인 BLOB형식으로 저장하고, 효율적인 공간연산을 위해 공간 데이터의 헤더는 비공간 데이터가 저장되는 데이터 페이지에 함께 저장이 된다.

4.3 공간 데이터 연산

4.3.1 공간 데이터 삽입연산

유령현상(Phantom Phenomenon)의 방지와 트랜잭션의 독립성을 보장하기 위해 삽입을 수행하기 전에 현재 레코드가 삽입될 위치에는 Commit Duration 배타잠금을 요청하고, 다음 레코드에 대해서는 Instant 배타 잠금을 요청한다. 잠금이 허락되면 공간 데이터를 헤더와 실제 점리스트를 분리한 후, 헤더에 있는 실제 점 리스트 데이터의 크기에 따라, 크기가 작으며 비공간 데이터와 함께 저장하고 Flag를 0으로 하며, 만약 크기가 작으면 BLOB형식으로 저장하고 Flag를 1로 한다.

알고리즘 1. 공간 데이터 삽입

입력: Transaction ID, Table OID, 비공간 레코드(ARecord), 공간 레코드(SRecord)

출력: Record ID

```
SNT_RID InsertSpatialRecord(TransID, TableOID, ARecord,
        SRecord)
{
  newRID = GetNextRecordID(TransID,TableOID);
  //비공간이 삽입될 새로운 RID를 얻는다.
  LockRecord(TransID,TableOID, newRID, COMMIT_
  DURATION);
  // 새로운 RID에 대해 Commit Duration 배타잠금을 요
  청한다.
  memcpy(SHead, SRecord, sizeof(SHead);
  sizeofVertex = sizeof(SRecord) - sizeof(SHead);
  memcpy(SVertex, SRecord+sizeof(SHead), sizeofVertex);
  // SRecord를 헤더와 점리스트로 분리한다.

  If (sizeofVertex > PAGESIZE ) {
    BLOBInsert(SVertex, &PID);
    Record.Flag = 1;        // BLOB 여부
    Record.ARecord = ARecord + SHead + PID;
    Insert(Record);
    //SRecord의 점리스트를 BLOB으로 저장하고 첫번째
    PID를 받는다.
    //Flag를 1로 하고 첫번째 PID와 SRecord의 헤더를
    비공간 레코드에 넣고 데이터 //페이지에 저장한다.
  }
  else {
    Record.Flag = 0;    // BLOB 여부
    Record.ARecord = ARecord + SRecord;
    Insert(Record);
    // Flag를 0으로 하고 비공간 데이터와 공간 데이터를
    같이 데이터 페이지에
    // 저장한다.
  }
  return newRID;
}
```

4.3.2 공간 데이터 갱신연산

먼저 공간 데이터가 속해 있는 비공간 데이터에 대해 Commit Duration 배타잠금을 요청하여 다른 트랜잭션이 접근한 데이터를 갱신하거나, 갱신한 트랜잭션이 Commit하기 전에 갱신한 데이터를 다른 트랜잭션이 접근하여 Cascade Rollback이 발생하는 것을 방지한다. 비공간 데이터에 대해서는 갱신이 기존의 레코드를 갱신하는 것이지만, 공간 데이터의 점리스트가 대용량 데이터일 경우에는 회복방법이 쉐도우 페이지 방법이기 때문에 기존의 데이터를 삭제하고, 새로운 데이터를 삽입하는 방법으로 이루어진다. 따라서 공간 데이터의 점리스트 크기가 페이지 이상일 때에는 새롭게 BLOB형식으로 데이터를 저장하고 그 BLOB의 첫 번째 페이지 식별자를 갱신 레코드에 기록한다. 그러나 기존에 저장된 공간 데이터 또한 대용량 데이터일 경우에는 기존의 저장된 BLOB에 대한 삭제연산을 트랜잭션의 Pending리스트에 추가한다.

알고리즘 2. 공간 데이터 갱신

입력: Transaction ID, Table OID, Record ID, ARecord, SRecord

출력: TRUE or FALSE

```
BOOL UpdateSpatialRecord(TransID, TableOID, RID,
      ARecord, SRecord)
{
  LockRecord(TransID,TableOID, RID, EXCLUSIVE_
  LOCK);
  // RID에 대해 배타잠금을 요청한다.
  memcpy(SHead, SRecord, sizeof(SHead);
```

```
  sizeofVertex = sizeof(SRecord) - sizeof(SHead);
  memcpy(SVertex, SRecord+sizeof(SHead), sizeof
  Vertex);
  // SRecord를 헤더와 점리스트로 분리한다.

  If (sizeofVertex > PAGESIZE ) {
    BLOBInsert(SVertex, &PID);
    Record.Flag = 1;          // BLOB 여부
    Record.ARecord = ARecord + SHead + PID;
    // BLOB으로 SRecord의 점리스트를 저장하고
    BLOB의 첫번째 페이지 식별자를 받는다

    BLOBDelete(PID);
    // BLOB의 첫번째 PID를 얻어서 트랜잭션의
    Pending리스트에 BLOB의 삭제
  else {
    Record.Flag = 0;          // BLOB 여부
    Record.ARecord = ARecord + SRecord;
    // Flag를 0으로 하고 ARecord과 SRecord가 같이 저
    장된 갱신 비공간 데이터 레코드를 생성한다.
  }
  UpdateRecord();
  // 갱신 비공간 데이터 레코드로 RID에 해당하는 비공
  간 데이터 레코드를 갱신한다.
  Return TRUE;
}
```

4.3.3 공간 데이터 삭제연산

데이터를 삭제하기 전에 유령현상(Phantom Phenomenon)을 방지하기 위해 현재 레코드와 다음 레코드에 대해 Commit Duration 배타잠금을 요청한다. 비공간 데이터의 삭제를 수행하기 전에 공간 데이터가 BLOB으로 저장되어 있는지 아니면 비공간 데이터와 같이 저장되었는지를 조사하여, BLOB으로 저장되어 있으면 BLOB에 대한 삭제연산을 트랜잭션의 Pending리스트에 추가한다. 다음으로 실제 비공간 데이터의 삭제를 수행한다.

알고리즘 3. 공간 데이터 갱신

입력: Transaction ID, Table OID, Record ID
출력: TRUE or FALSE

```
BOOL DeleteSpatialRecord(TransID, TableOID, RID)
{
  LockRecord(TransID, TableOID, RID, COMMIT_
  DURATION);
  // 현재 레코드와 다음 레코드에 대해 Commit Duration
  배타잠금을 요청.
  If ( Record.Flag == 1 ) {
    BLOBDelete(RID);
    // 비공간 데이터의 BLOB의 첫번째 PID로 BLOB의
    삭제연산을 트랜잭션 Pending 리스트에 추가한다.
  }
  Delete(RID);
  // 비공간 데이터 레코드의 삭제한다.
  Return TRUE;
}
```

4.3.4 공간 데이터 읽기 연산

트랜잭션의 격리레벨에 따라 잠금정책이 달라진다. 만약 트랜잭션의 격리레벨이 Dirty Read이면 현재 읽어들일 데이터에 대해 잠금을 요청하지 않고 데이터를 읽어들인다. 만약 Commit Read라면 현재 읽어 들일 데이터에 대해 Instant Duration 공유잠금을 요청한다. 그러나 트랜잭션의 격리레벨이 Repeatable Read이상이면 레코드에 대해 Commit Duration 공유잠금을 요청한다. 트랜잭션에 격리레벨에 따라 잠금을 요청하였다면 데이터를 읽어들인다.

알고리즘 4. 공간 데이터 읽기

입력: Transaction ID, Transaction ISO Level, Table OID, Record ID,
출력: TRUE or FALSE

```
BOOL ReadSpatialRecord(TransID, TransISOLevel,
        TableOID, RID)
{
  If (TransISOLevel == COMMIT_READ) {
    LockRecord(TransID, TableOID, RID, INSTANT_
    DURATION);
    // 현재 레코드에 대해 Instant Duration 공유잠금을
    요청한다.
  }
  else {
    LockRecord(TransID, TableOID, RID, COMMIT_
    DURATION);
    // 현재 레코드에 대해 Commit Duration 공유잠금을
    요청한다.
  }
  If (Record.Flag == 1) {
    BLOBRead(PID, &SVertex);
    // 공간 데이터의 BLOB의 첫번째 페이지 식별자를
    이용하여 데이터를 읽어들인다.
  }
```

```
    Read(RID,&Record);
    // 비공간 데이터를 읽어들인다.
    Return TRUE;
}
```

4.4 실험 및 평가

본 절에서는 제안된 공간 저장 기법의 성능을 비교하기 위해 기존의 대용량 데이터 저장 기법인 모든 공간 데이터에 대해 균일한 페이지 할당 정책을 하는 시스템을 구현하고 이를 본 제안 시스템과의 실험 평가를 기술한다. 시스템은 각각 한 페이지의 크기를 4K 바이트로 고정하고, 데이터의 크기가 1K 바이트 미만인 스몰(small) 데이터와 4K 바이트 이상인 라지(large) 데이터로 구분하여 각각의 구성 비율을 변화시키면서 성능을 비교 평가한다.

다음 그림 11, 그림 12, 그림 13은 각각 삽입, 읽기, 삭제 수행 비용의 비교 그래프로 1번이 기존의 대용량 데이터 저장 기법으로 구현된 시스템의 응답 결과이고 2번이 제안 시스템의 응답 결과로 단순한 점이나 심볼과 같은 적은 용량의 스몰 공간 데이터가 많은 경우 성능이 월등히 나아짐을 알 수 있다.

평가 결과 스몰 데이터의 비율이 높을수록 제안기법의 성능이 향상되었으며, 삽입은 최대 80%, 삭제는 최대 84%, 읽기는 최대 83%의 성능향상이 있었다. 또한 스몰 데이터의 비율이 0%이더라도 기존의 대용량 데이터 저장기법과 각 연산이 비용 큰 차이가 없음을 알 수 있다. 따라서 제안 기법은 스몰 데이터에 대해 빠른 연산을 수행하며, 대용량 데이터 또한 효율적으로 처리함을 본 실험 평가를 통하여 알 수 있다.
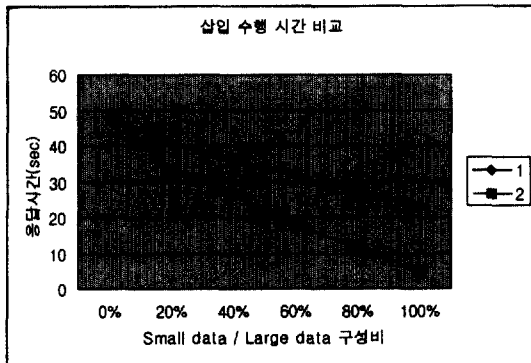
그림 11. 삽입 수행 시간 비교
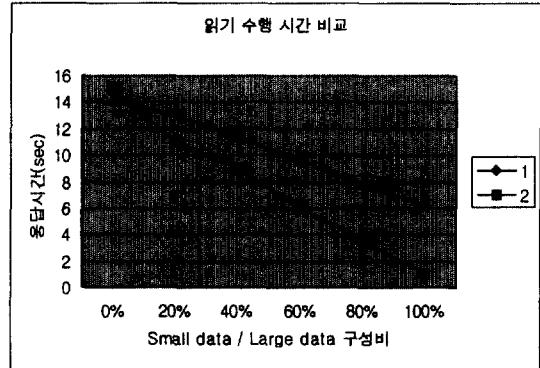
그림 12. 읽기 수행 시간 비교
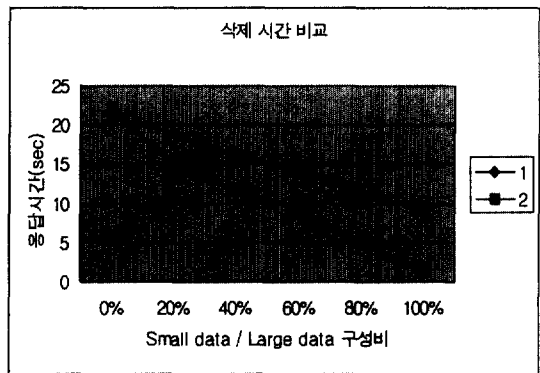
그림 13. 삭제 수행 시간 비교

5. 결 론

기존의 대용량 데이터 저장관리자들은 데이터의 크기가 대용량이라는 특성에 기반하여 설계 및 구현되었다. 따라서 공간 데이터와 같이 기존의 멀티미디어 데이터와 달리 레이어 단위로 데이터의 크기가 일정하며, 데이터의 크기 또한 몇 바이트에서 최대 몇 테라까지 다양하다는 데이터 특성을 가지고 있으면 기존에 제시된 멀티미디어 데이터 저장관리기법으로는 디스크 공간 낭비 및 시스템의 성능저하라는 문제를 가지게 된다. 따라서 본 논문에서는 이러한 공간 데이터의 특성을 고려한 공간 데이터 저장기법을 제시하였다.

제안된 기법은 첫 번째로 공간 데이터에 대해 사용자가 질의를 수행시 고비용의 공간연산이 수행된다. 이러한 연산비용을 줄이기 위해 여과과정을 수행하는데 이때 여과과정시 공간 데이터의 헤더의 접근을 필요로 한다. 또한 공간 데이터의 색인인 R-Tree

를 구성시 각 공간 데이터의 최소경계사각형을 필요로 한다. 따라서 공간 데이터의 크기가 대용량일 때, 무조건 대용량 저장 데이터 관리기법인 BLOB형식으로 저장할 시 많은 디스크 입출력이 발생한다. 따라서 본 제안기법은 공간 데이터의 헤더와 점리스트 부분을 분리하여 공간 데이터의 헤더는 항상 비공간 데이터 레코드와 함께 저장하고, 실제 점리스트는 크기에 따라 크면 대용량 저장 데이터 관리기법인 BLOB형식으로 데이터를 저장하고 작으면 비공간 데이터 레코드와 함께 저장한다.

두 번째로 공간 데이터의 크기특성인 레이어 단위로 크기가 비슷하게 나타나며, 데이터의 크기 또한 몇 바이트에서 최대 몇 테라까지 다양하다는 데이터의 특성에 기반하여 크기에 따라 공간 데이터의 점리스트의 저장 위치를 다르게 하였다. 공간 데이터의 점리스트의 페이지보다 작으면 비공간 데이터 레코드와 같이 저장이 되며, 공간 데이터 크기가 크면 따로 순차접근이 용이하고, 무제한의 크기를 지원하는 대용량 저장 데이터 관리기법인 BLOB형식으로 데이터를 저장한다.

세 번째로 제안기법을 저장관리자에서 이용하기 위해서는 동시성 제어 및 복구가 필수적이다. 따라서 본 제안기법은 ARIES에 기반하여 크기가 작은 데이터에 대해서는 로깅에 기반하여 회복을 수행하였으며, 크기가 큰 대용량 데이터에 대해서는 쉐도우 페이지기법에 기반하여 회복을 수행하였다. 동시성 제어는 엄정 잠금 2단계에 기반한 잠금기법을 이용하였고, 공간 데이터의 접근패턴을 일정하게 하여 따로 공간 데이터에 대한 잠금이 필요 없기 때문에 잠금의 부하가 적다.

본 제안기법은 기존의 대용량 데이터 저장관리기법인 BLOB을 그대로 사용하였을 경우 보다 스몰 데이터의 양에 따라 최대 80%정도의 성능향상을 보였다.

향후 연구과제로는 공간 저장 관리자에서 고비용의 공간연산에 대해 비용 절감 기법이 고려되어야 한다.

참 고 문 헌

[1] Ralf Hartmut Guting,: An Introduction to Spatial Database Systems, *VLDB Journal 3(4)*, pp.357-399, 1994.

[2] M. J. Carey, et al, Object and File Management in the EXODUS Extensible Database System, Proc. Of 12 th VLDB, pp.375-383, 1986.

[3] EXODUS Project Group, Using the EXODUS Storage System V3.1 *EXODUS Project Document*, University of Wisconsin-Madison, 1993.

[4] T. J. Lehman., B. G. Lindsay, The Starburst Long Field Manager, *Proc. of 15th VLDB*, pp. 276-285, 1989.

[5] T. Brinkhoff., H.P.Kriegel, and B.Seeeger, Parallel Processing of Spatial Joins Using R-Trees, Processings of 12th International Conference on Data Engineering(ICDE'96), New Orleans, LA, 1996.

[6] Mohan, C., Haderle. D., Lindsay, B., Pirahesh. H., Schwarz. P., ARIES, A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging, ACM Transaction on Database Systems. January 1989.

[7] M. Stonebraker., et al., The SEQUOIA 2000 Project, Proc. Of 3rd Symposium of Spatial Database '93. pp.397-412, 1993.

[8] Mohan, C., ARIES/KVL: A Key-Value Locing Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes, Proc. 16th International Conference on Very Large Data Bases, Brisbane, August 1990.

[9] Mohan. C., ARIES/IM: An Efficient and High Concurrency Index Management Method using Write-Ahead Logging, IBM Research Report RJ6846, IBM Almaden Research Center, August 1989.

[10] Mohan, C., Pirahesh, H., ARIES-RRH: Restricted Repeating of History in the ARIES Transaction Recovery Method, Proc. 7th International Conference on Data Engineering, Kobe, April 1991.

[11] Mohan, C., Narang, I., Recovery and Coherency-Control Protocols for Fast Intersystem Page

Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment, Proc. 17th International Conference on Very Large Data Bases, Barcelona, September 1991.
[12] Rothermel, K., Mohan, C. : ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions, Proc. 15th International Conference on Very Large Data Bases, Amsterdam, August 1989.

김 종 훈

1991년 인하대학교 전자계산공학과(공학사)
1993년 인하대학교 대학원 전자계산공학과(공학석사)
1998년 인하대학교 대학원 전자계산공학과(공학박사)
1996년 인하대학교 전자계산공학과 전임대우
1998년~현재 (주)케이지아이 대표이사
관심분야 : 데이터베이스, 멀티미디어 데이터베이스 시스템, 지리 정보 시스템, SAN

정 현 민

1984년 연세대학교전자공학과졸업(공학사)
1986년 연세대학교 본대학원 전자공학과(공학석사)
1986년~1992년 한국통신 연구개발본부
1996년 연세대학교 본대학원 전자공학과(공학박사)
1996년~현재 한국통신 가입자망연구소 무선망설계연구실장
관심분야 : 무선망설계 시스템 개발/엔지니어링, 영상부호화

장 성 인

1985년 경북대학교 전자계산학과 졸업 (이학사)
1988년 아주대학교 대학원 전자계산학과 졸업(이학석사)
1991년~현재 한국통신 연구개발본부 가입자망연구소 선임연구원
관심분야 : 대용량 데이터베이스 시스팀, 멀티미디어 데이터베이스 시스팀, 시스팀 성능 분석

정 미 영

1993년 덕성여자대학교 전산학과 졸업 (이학사)
1996년 연세대학교 전산과학과 대학원 졸업 (이학석사)
1996년~현재 한국통신 연구개발본부 가입자망연구소 무선망설계연구실 전임연구원
관심분야 : 무선망 설계 시스팀, 지형데이터 처리 시스팀, 데이터베이스시스팀, 데이터마이닝, GML, 컴포넌트기반 소프트웨어 공학